Special Copyright Notice

ANSI/AIAA
G-031-1992

Guide

Life Cycle Development of Knowledge Based Systems Using DoD-Std 2167A

ANSI

AIAA

ANSI/AIAA
G-031-1992

American National Standard

Guide for Life Cycle Development of Knowledge Based Systems with Dod-Std-2167A

Sponsor

American Institute of Aeronautics and Astronautics

Approved July 9, 1993

American National Standards Institute

Abstract

The purpose of this guide is to provide program managers and system engineers with a blueprint for developing and managing knowledge based systems within typical aerospace environments. The guide is consistent with the requirements of Dod-Std-2167A, yet permits the creativity needed for the still maturing field of artificial intelligence and knowledge based systems.

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria have been met by the standards developer.

Consensus is established when, in the judgement of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken to affirm, revise, or withdraw this standard no later than five years from the date of approval. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

Guide for life cycle development of knowledge based systems with
DoD-Std-2167A / sponsor, American Institute of Aeronautics and
Astronautics; approved, American National Standards Institute.
p. cm.
At head of title: American national standard.
"ANSI/AIAA G-031-1992"
ISBN 1-56347-025-X
1. Computer software--Development--Standards--United States.
2. Expert systems (Computer science)--Standards--United States.
I. American Institute of Aeronautics and Astronautics. II. American
National Standards Institute. III. Title: American national
standard guide for life cycle development of knowledge based systems
with Dod-Std 2167A.
QA76.76.D47G85 1993
623--dc20 92-46900
CIP

Published by
American Institute of Aeronautics and Astronautics
370 L'Enfant Promenade, SW, Washington, DC 20024

Printed in the United States of America

CONTENTS

Figures

Foreword

This Guide for Life Cycle Development of Knowledge Based Systems with DoD-Std-2167A has been sponsored by the American Institute of Aeronautics and Astronautics as part of its Standards Program.

Knowledge Based Systems (KBS) have been emerging out of the artificial intelligence (AI) research laboratories and into the main stream of aerospace software. As this evolution occurs, the methodologies and constraints of software development need to be overlaid onto those of KBS development. This overlay is considered necessary in order for KBS to gain wide acceptance and use in aerospace systems.

The purpose of the Guide is to provide program managers and system engineers within a blueprint for developing and managing KBS within typical aerospace environments. It is published as a guide at this point, instead of as a full standard, because it was felt that the field is not mature enough and flexibility is still needed.

The AIAA Standards Procedures provide that all approved Standards, Recommended Practices, and Guides are advisory only. Their use by anyone engaged in industry or trade is entirely voluntary. There is no prior agreement to adhere to any AIAA standards publication and no commitment to conform to or be guided by any standards report.

In formulating, revising, and approving standards publications, the Committees on Standards will not consider patents which may apply to the subject matter. Prospective users of the publication are responsible for protecting themselves against liability for infringement of patents or copyrights, or both.

This project is the undertaking of the AIAA Committee on Standards for Artificial Intelligence (AI/CoS) and its Life Cycle Working Group (LCWG). The LCWG developed this document and incorporated comments of reviewers from government, industry, and academia. The following people played key roles in bringing this Guide to publication and held the noted positions at the time of their efforts:

Peter A. Kiss (Sentar, Inc.) (LCWG Lead)
Michael Freeman (NASA SSF Program Office) (AI CoS Co-Chair)
Charles Hall (Lockheed Research Center) (AI CoS Co-Chair)
Jack Aldridge (McDonnell Douglas)
Peter Russo (Anderson Consulting)

Suggestions and improvement to this document are encouraged. They should be sent to:

AIAA Standards Department
370 L'Enfant Promenade, SW
Washington, DC 20024-2518

The AIAA Standards Technical Council (A. H. Ghovanlou, Chairman) approved the document in November 1992.

1.0 INTRODUCTION

1.1 Background

During the past ten years, the Knowledge Based Systems (KBS) branch of Artificial Intelligence (AI) has matured considerably. Many small and medium sized prototype systems have been successfully developed and implemented. The few existing larger KBS are much more complex and have been implemented at a slower rate. The organizations at the leading edge of using AI, ones that have been developing KBS and applying them, are moving toward the integration of KBS into the mainstream of their computing environments. This move is leading to a more traditional total systems approach to KBS, making them an integral part of the system instead of a stand alone tool or application. With the emphasis shifting to a systems approach, comes the need for more rigorous development and integration methodologies. This trend, coupled with the general aerospace community's desire to control costs and schedules, provides the impetus for having a KBS life cycle model.

Several basic phases are inherent parts of any software (including AI / KBS) development program. These are: problem conceptualization / definition; system design; system development; testing and integration; and maintenance and enhancement. The sequence in which these are carried out, the amount of emphasis/effort given each phase, and the controls associated with the execution of work combine to define a life cycle model. At present, there are several generally accepted models for software development. These are the waterfall, the spiral, and the evolutionary (rapid prototyping) models. There are many variations of these three as well as some lesser known models. The best known is the waterfall model typically used with DoD-Std-2167A. This has been used (in one form or another) for the development of thousands of software systems, and is being mandated for nearly all DoD development of software. Many of the existing software development models have been tried on KBS with varying degrees of rigor, and success. Each has strengths and shortfalls in relation to KBS development under the emerging, more formal conditions discussed above. It is to meet that emerging demand (i.e., integration into general software engineering methodology), and to provide some structure and uniformity to aerospace knowledge based system development, that this Guide is written.

1.2 Purpose

The purpose of this guide is to provide program managers (both in government and industry) and system engineers with a blueprint for developing and managing KBS within typical aerospace environments. Details are included on the various phases of KBS life cycles and how they fit within a waterfall software development life cycle exemplified by the DoD-Std-2167A framework. Reviews to be held during development are discussed along with highlights of recommended documentation. The detailed contents of KBS documentation are not discussed in this guide.

1.3 Applications

As presented, this guide is focused on the development of KBS during what is referred to as the *Full Scale Development* of a system (i.e., the final version before proceeding with production). With very little tailoring, it can also be used for KBS development during the earlier phases of *Concept Definition* and *Demonstration / Validation* of systems.

The presentation in this guide uses the DoD-Std-2167A framework with the waterfall model for conventional software development. Naturally, the guide can be easily adapted to any waterfall type model implementation.

Finally, although the presentation is limited to Knowledge Based Systems because they are at the leading edge of the state-of-the-application of AI, much of the guide can be used for other branches of AI and possibly for other areas where critical methods need to be prototyped within a larger system context.

2.0 REFERENCED PUBLICATIONS

The following documents are applicable when using this guide for the development of KBS.

1. Dod-Std-2167A Defense System Software Development; 29 February 1988
2. Dod-HDBK-287 A Tailoring Guide for Dod-Std-2167A, Defense System Software Development; (Draft) 14 November 1988
3. Mil-Std-1521B Technical Reviews and Audits for Systems, Equipment, and Computer Software
4. IEEE Std-729-1983 Standard Glossary of Software Engineering Terminology

3.0 OVERVIEW OF DOD-STD-2167A AND OTHER LIFE CYCLE MODELS

3.1 Introduction

The general purpose of software development life cycle models is to introduce discipline into the development process and thus reduce risk. This discipline, in turn, leads to better end products and better management of the development process. Readers who are well versed in software life cycles may choose to skip chapter 3.0 and move on to KBS life cycles.

The most prevalent software life cycle model to date has been the waterfall model. Variations of the waterfall have been used for over 20 years. The Department of Defense (DoD) has adapted this model officially, in the DoD-Std-2167A document, and mandates its use on DoD software. It should be noted that the DoD-Std-2167A allows for tailorings divergent from a waterfall model. Since other versions of the waterfall model can be translated to the one in the DoD Standard, we will use DoD-Std-2167A here as an example.

Other software models have been used successfully for various applications. These include the spiral model and the evolutionary model, which are summarized briefly in section 3.3.

3.2 DoD-Std-2167A Overview

The purpose of the Standard is to establish requirements to be applied during the acquisition, development, or support of software systems being acquired by DoD. The Standard primarily applies to Computer Software Configuration Items (CSCI), but can be used for non-CSCI items such as support software and firmware.

3.2.1 General Requirements

In addition to providing a detailed waterfall model, 2167A provides guidance on overall management. This guidance is in the form of general requirements and encompasses the areas discussed below.

3.2.1.1 Software Development Management. Implement a process for managing the development of the deliverable software. This includes the use of the model phases discussed in section 3.2.2 and the documentation of section 3.2.3. In addition to these, the management of software will include: software development planning; risk management; management reviews; security; subcontract management; interface with independent verification and validation agents; maintenance of a software development library; implementation of a corrective action process; and a problem/change reporting activity.

3.2.1.2 Software Engineering. Perform software engineering that incorporates the following good practices: utilize systematic software development methods; provide a software engineering environment for the development phase; perform safety analysis to reduce hazards; consider incorporation of non-developmental software; organize the software systematically into CSCIs and decompose to Computer Software Components (CSCs) and Computer Software Units (CSUs); provide traceability of requirements to design; use Higher Order Languages (HOL) whenever possible; use design and coding standards specified; maintain software development files for each CSU; analyze processing resources and reserves to assure proper operations.

3.2.1.3 Formal Qualifications Testing (FQT). Conduct FQT of each CSCI on the target computer system or equivalent. This shall include stressing the software at the limits of its specified requirements. Included under Formal Qualification Testing are: FQT planning; the necessary software test environment; independence of FQT activities; and traceability of requirements to the test cases.

3.2.1.4 Software Product Evaluations. Conduct evaluations of deliverable software and documentation to include: independent product evaluations; a final (acceptance) evaluation to show the product meets the requirements; prepare and maintain records of the evaluations; and the recorded criteria against which the software was evaluated.

3.2.1.5 Software Configuration Management. Perform software configuration management in compliance with the following: perform configuration item identification in accordance with identification scheme specified in contract; implement configuration control on identified items; provide a configuration status accounting system; provide storage, handling and delivery of project media; provide for implementation of Engineering Change Proposals (ECP) and Specification Change Notices (SCN).

3.2.1.6 Transition to Software Support. Provide transition support in compliance with the following requirements: deliverable code shall be capable of being regenerated and maintainable; prepare plans for transitioning from development to support; perform installation and checkout of delivered software; provide software support and documentation for delivered software.

The above provides a quick summary of the general requirements. For further details, the reader should consult DoD-Std-2167A. Most of these requirements are applied to each phase of the software life cycle and many of them are applicable to Knowledge Based Systems.

3.2.2 Phases of a Waterfall Model

The discussion below is for software development during *Full Scale Development* (FSD) of a system. As such, the previous phases of *Concept Definition* and *Demonstration / Validation* have established some of the foundations for the activity. System analysis / definition has documented the preliminary concepts, their feasibility and cost / benefit tradeoffs. The traditional phases of a waterfall model during FSD are shown in Figure 3-1. These phases are briefly summarized below.

3.2.2.1 System Requirements Analysis / Design. During this phase, the contractor analyzes the preliminary system specification and determines whether the software requirements are consistent and complete. Analysis is conducted to determine the best allocation of system requirements between hardware, software, and personnel in order to partition the system into HWCIs, CSCIs, and manual operations. As a result, a preliminary set of engineering requirements and associated sets of external interfaces are defined for each CSCI.

3.2.2.2 Software Requirements Analysis. During this phase, a complete set of software engineering requirements and interfaces are defined for each CSCI.

3.2.2.3 Preliminary Design. The purpose of this phase is to develop the design which includes mathematical models, functional flows, and data flows. For each CSCI, preliminary design is developed and software and interface requirements are allocated to CSCs. Also, the requirements for testing and integration of CSCs are established, to include stressing the software.

3.2.2.4 Detailed Design. During this phase, the design is refined to the extent that it is ready for coding. For each CSCI, the detailed design is developed, and the requirements of each CSC are allocated down to CSUs. Detailed designs of the external interfaces of each CSCI are developed and

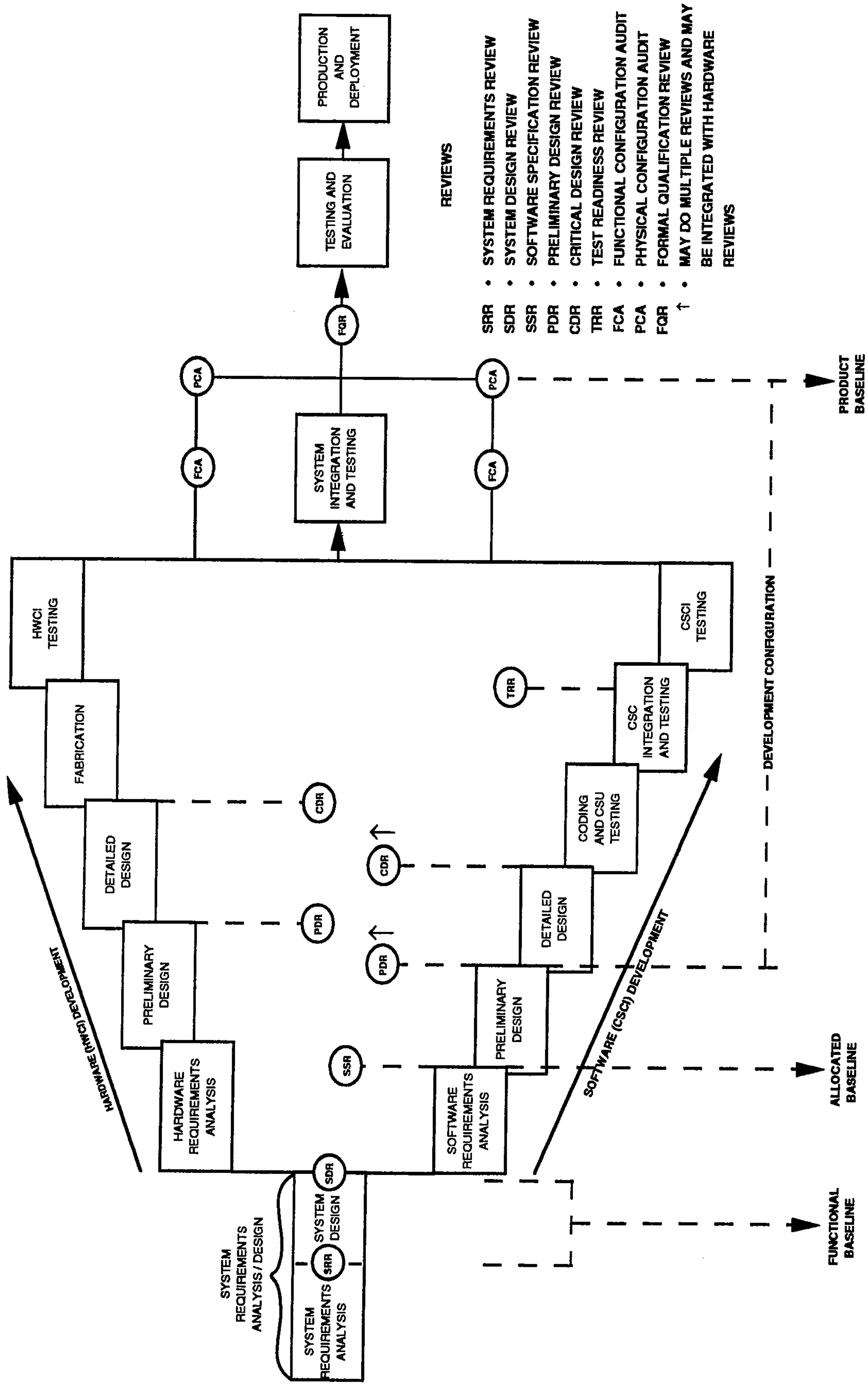


Figure 3-1 An Example of System Development Phases, Reviews, and Audits

allocated. CSC and CSU test requirements, responsibilities and test cases are developed and documented.

3.2.2.5 Coding and CSU Testing. During this phase, the CSUs are coded and tested, according to developed procedures, to ensure that the logic and algorithms are correct and the CSUs satisfy their requirements. Based on testing results, the necessary revisions are made to the design, code and documentation to reflect corrections. Also, test procedures are developed for conducting CSC tests.

3.2.2.6 CSC Integration and Testing. In this phase the CSUs are integrated into CSCs and tested to ensure that CSC algorithms and logic are functioning properly and that requirements are satisfied. The necessary revisions are made to design, code and documentation to reflect changes based on test results.

3.2.2.7 CSCI Testing. In this phase, the CSCs are integrated to form the CSCIs and the full capabilities of the CSCIs are tested against requirements. The CSCI interfaces are also tested and revisions are made to code, design and interface documents to reflect testing results.

3.2.2.8 System Integration and Testing. This phase concludes software development and testing. As the system is tested, the necessary revisions are made to software code, design and documentation.

3.2.2.9 Operation and Maintenance (production and deployment). Once the system passes Formal Qualification Reviews (FQR), it enters the next portion of its life cycle, production or deployment. During this phase (deployment), the system is operated and maintained to carry out its intended functions.

The above phases are representative of the activity progression under a waterfall model. Some versions may consolidate several phases into one or rename them, but the sequence of events is constant across models. Full details of the activities of each phase are in the DoD-Std-2167A for reference.

3.2.3 Technical Reviews and Documentation

Section 3.2.2 summarized the phases of the waterfall model. In this section the major technical / milestone reviews and program documentation associated with software development (per 2167A) are summarized. Figure 3-2 depicts the relationship between development phases, reviews and associated documentation.

3.2.3.1 Technical Reviews. The following are taken from Mil-Std-1521B as summaries for normal reviews associated with a software development effort. They probably constitute the maximal set of reviews that may be required for a given project.

3.2.3.1.1 System Requirements Review (SRR). The objective of this review is to ascertain the adequacy of the contractor's efforts in defining system requirements. It will be conducted when a significant portion of the system functional requirements has been established.

3.2.3.1.2 System Design Review (SDR). This review shall be conducted to evaluate the optimization, correlation, completeness, and risks associated with the allocated technical requirements. Also included is a summary review of the system engineering process which produced the allocated technical requirements and of the engineering planning for the next phase of effort. Basic development/coding/manufacturing considerations will be reviewed and planning for quality/production engineering in subsequent phases will be addressed. This review will be conducted when the system definition effort has proceeded to the point where system characteristics are defined and the configuration items are identified.

3.2.3.1.3 Software Specification Review (SSR). This is a review of the finalized Computer Software Configuration

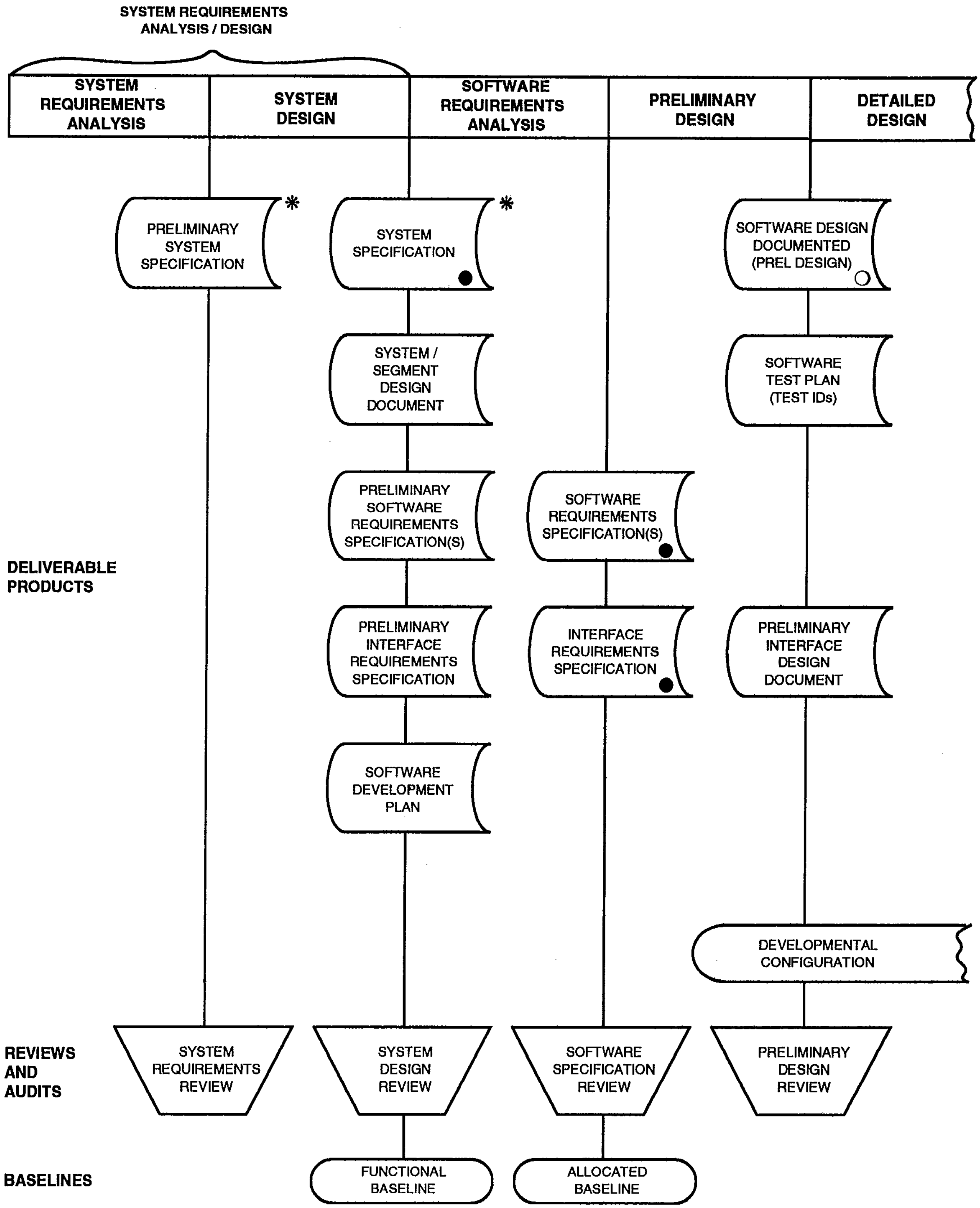


Figure 3-2 Deliverable Products, Reviews, Audits, and Baselines

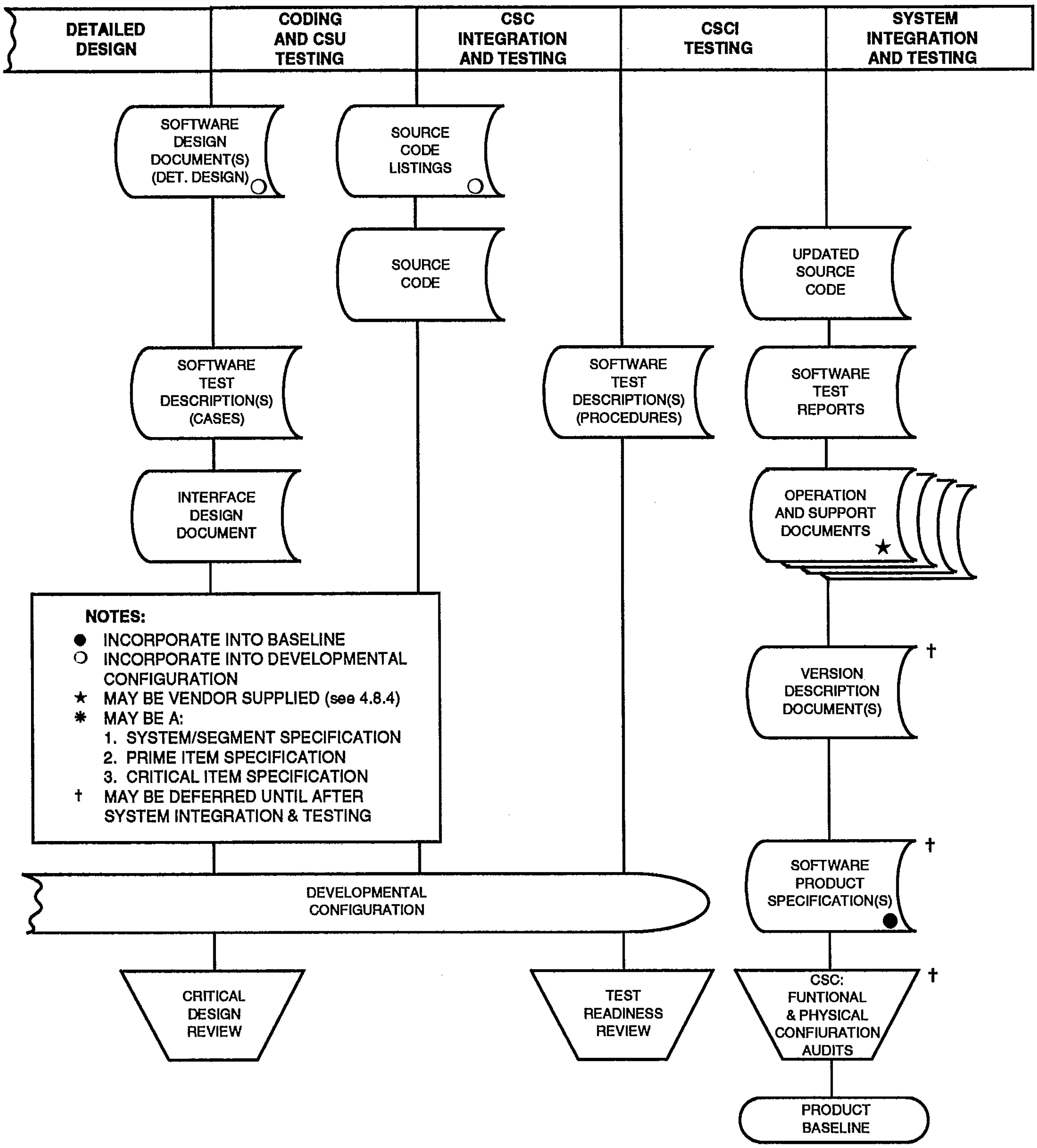


Figure 3-2 Deliverable Products, Reviews, Audits, and Baselines - continued

Item (CSCI) requirements and operational concept. The SSR is conducted when the CSCI requirements have been sufficiently defined to evaluate the contractor's responsiveness to and interpretation of the system, segment, or prime item level requirements. A successful SSR is predicated upon the contracting agency's determination that the Software Requirements Specification, Interface Requirements Specification(s), and Operational Concept Document form a satisfactory basis for proceeding into preliminary software design.

3.2.3.1.4 Preliminary Design Review (PDR). This review shall be conducted for each configuration item or aggregate of configuration items to:

(1) evaluate the progress, technical adequacy, and risk resolution (on a technical, cost, and schedule basis) of the selected design approach;

(2) determine its compatibility with performance and engineering specialty requirements of the Hardware Configuration Item (HWCI) development specification;

(3) evaluate the degree of definition and assess the technical risk associated with the selected manufacturing methods/processes; and

(4) establish the existence and compatibility of the physical and functional interfaces among the configuration item and other items of equipment, facilities, computer software, and personnel.

For CSCIs, this review will focus on:

(1) the evaluation of the progress, consistency, and technical adequacy of the selected top-level design and test approach;

(2) the compatibility between software requirements and preliminary design; and

(3) the preliminary version of the operation and support documents.

3.2.3.1.5 Critical Design Review (CDR). This review shall be conducted for each configuration item when the detail design is essentially complete. For CSCIs, this review will focus on the determination of the acceptability of the detailed design, performance, and test characteristics of the design solution, and on the adequacy of the operation and support documents. The purpose of this review will be to:

(1) determine that the detail design of the configuration item under review satisfies the performance and engineering specialty requirements of the HWCI development specifications;

(2) establish the compatibility among the configuration item and other items of equipment, facilities, computer software and personnel;

(3) assess configuration item risk areas (on a technical, cost and schedule basis);

(4) assess the results of the producibility analyses conducted on system hardware; and

(5) review the preliminary hardware product specifications.

3.2.3.1.6 Test Readiness Review (TRR). A review conducted for each CSCI to determine whether the software test procedures are complete and to assure that the contractor is prepared for formal CSCI testing. Software test procedures are evaluated for compliance with software test plans and descriptions, and for adequacy in accomplishing test requirements. At TRR, the contracting agency also reviews the results of informal software testing and any updates to the operation and support documents. A successful TRR is predicated on the contracting agency's determination that the software test procedures and informal test results form a satisfactory basis for proceeding into formal CSCI testing.

3.2.3.1.7 Functional Configuration Audit (FCA). A formal audit to validate that the development of a configuration item has been completed satisfactorily and that the configuration item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification. In addition, the completed operation and support documents shall be reviewed.

3.2.3.1.8 Physical Configuration Audit (PCA). A technical examination of a designated configuration item to verify that the configuration item "As Built" conforms to the technical documentation which defines the configuration item.

3.2.3.1.9 Formal Qualification Review (FQR). The test, inspection, or analytical process by which a group of configuration items comprising the system are verified to have met specific contracting agency contractual performance requirements (specifications or equivalent). This review does not apply to hardware or software requirements verified at FCA for the individual configuration item.

3.2.3.1.10 Production Readiness Review (PRR). This review is intended to determine the status of completion of the specific actions which must be satisfactorily accomplished prior to executing a production go-ahead decision. Its major emphasis is on hardware, but software is reviewed in its appropriate context. The review is accomplished in an incremental fashion during the Full-Scale Development phase, usually two initial reviews and one final review to assess the risk in exercising the production go-ahead decision. In its earlier stages the PRR concerns itself with gross level manufacturing concerns such as the need for identifying high risk / low yield manufacturing processes or materials or the requirement for manufacturing development effort to satisfy design requirements. The reviews become more refined as the design matures, dealing with such concerns as production planning, facilities allocation, incorporation of producibility-oriented changes, identification and fabrication of tools / test equipment, and long lead item acquisition. Timing of the incremental PRRs is a function of program posture and is not specifically locked into other reviews.

3.2.3.2 Software Development Documentation. Figure 3-2 depicts the documents recommended by DoD-Std-2167A. The excerpts below are from the various Data Item Description (DID) documents associated with them.

3.2.3.2.1 System / Segment Specification (SSS, #DI-CMAN-80008A). This document specifies the requirements for a system or a segment of a system. Upon Government approval and authentication, the SSS becomes the Functional Baseline for the system or segment. This document provides a general overview that may be used by training personnel, support personnel, or users of the system.

3.2.3.2.2 System / Segment Design Document (SSDD, DI-CMAN-80534) The SSDD describes the design of a system/segment and its operational and support environments. It describes the organization of the system as composed of HWCI, CSCI and manual operations. The SSDD contains the highest level of design information for the system and is used as a basis for developing lower level documents.

3.2.3.2.3 Software Requirement Specification (SRS, DI-MCCR-80025A). The SRS specifies the engineering and qualification requirements for a CSCI and is used as a basis for its design and formal testing. The SRS becomes the baseline of requirements against which the CSCI performance is assessed.

3.2.3.2.4 Interface Requirements Specification (IRS, DI-MCCR-80026A). The IRS specifies the requirements for the interfaces between the CSCIs as well as other configuration items. The implementation of interfaces is later assessed and checked for compliance with the IRS.

3.2.3.2.5 Software Development Plan (SDP, DI-MCCR-80030A). The SDP describes a contractor's plans for conducting software development. It covers the organizational responsibilities, the procedures, the management and the contract work breakdown for the effort.

3.2.3.2.6 Software Design Document (SDD, DI-MCCR-80012A). The SDD describes the complete design of a CSCI. It describes the allocation of requirements to CSCs and CSUs. The SDD is used to present the software design at PDR and CDR and is the basis for coding the software.

3.2.3.2.7 Software Test Plan (STP, DI-MCCR-80014A). The STP describes the formal qualification test plans for one or more CSCIs. It identifies the software test environment resources required for formal qualification testing and provides schedules for FQT activities. In addition, the STP identifies the individual tests that shall be performed during FQT.

3.2.3.2.8 Interface Design Document (IDD, DI-MCCR-80027A). The IDD specifies the detailed design for interfaces between CSCIs as well as to other configuration items. The IDD and IRS together serve to communicate and control interface design decisions. The IDD is used as the basis for software design of the interfaces. It in turn should reflect the requirements of the IRS.

3.2.3.2.9 Software Test Description (STD, DI-MCCR-80015A). The STD contains the test cases and test procedures necessary to perform qualification testing of CSCIs identified in the STP.

3.2.3.2.10 Source Code and Source Code Listing. This is the actual code and its listing as developed from the SDD. The executable object code produced from the source code is what is actually executed on the computers.

3.2.3.2.11 Software Test Reports (STR, DI-MCCR-80017A). The STR is a record of the formal qualification testing performed on a CSCI. It provides the government with a permanent record of FQT and can be used for retests as needed.

3.2.3.2.12 Version Description Document (VDD, DI-MCCR-80013A. The VDD identifies and describes a version of a CSCI or interim changes to previously released versions. It records data pertinent to the status and usage of a CSCI version or interim change.

3.2.3.2.13 Software Product Specification (SPS, DI-MCCR-80029A). The SPS consists of the SDD and the source code listing for a CSCI. Upon government approval and authentication following a Physical Configuration Audit (PCA), the SPS establishes the product baseline for the CSCI.

3.2.3.2.14 Operations and Support Documents. These consist of the documents delivered with the operational software to support system operations. They include the following: Computer Systems Operator's Manual (CSOM, DI-MCCR-80018A), Software User's Manual (SUM, DI-MCCR-80019A), Software Programmer's Manual (SPM, DI-MCCR-80021A), Firmware Support Manual (FSM, DI-MCCR-80022A), and Computer Resource Integrated Support Document (CRISD, DI-MCCR-80021A).

The details on how to develop the content and format of the above documents are contained in the referenced Data Item Description documents.

3.3 Other Software Life Cycle Models

Two other software life cycle models need to be discussed. These are the spiral model and the evolutionary model. The spiral model of software development was pioneered by Barry Boehm at TRW. Its distinguishing feature is a risk driven approach to the software process, rather than the specification driven one of the waterfall model. The evolutionary model, on the other hand, comes from the research and development community and is prototype driven. These two models are briefly summarized below.

3.3.1 A Spiral Model for Software Development

The spiral model shown in Figure 3-3 has evolved from the waterfall model through experience with government projects. The radial dimensions represents the cumulative cost incurred in development steps to date. The angular dimension represents progress made in completing each cycle of the spiral.

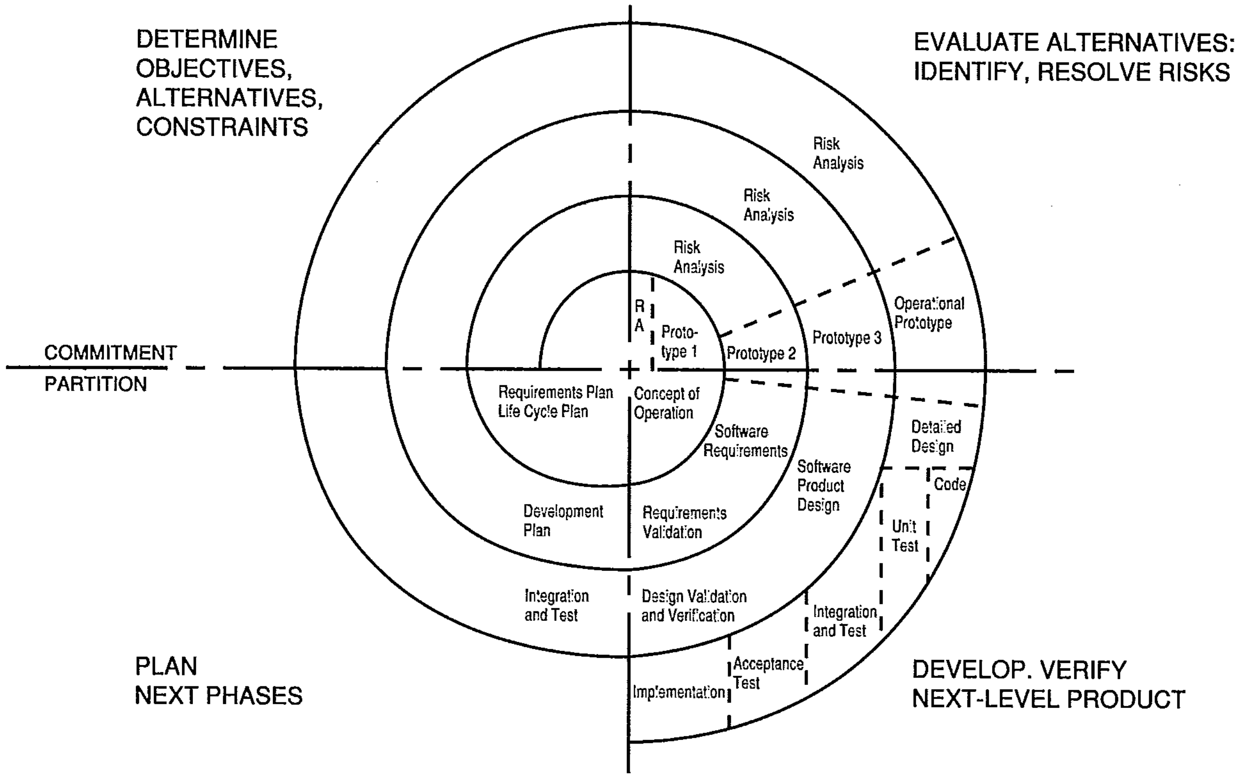


Figure 3-3 A Spiral Model for Software Development

The cycles of the spiral begin with the identification of objectives, alternatives and constraints. Next, the alternatives are evaluated with respect to the objectives and constraints. Areas of project risk are thus brought to light and focused upon. Based on the assessments, the area(s) selected is prototyped, designed, developed and tested. Having completed the cycle for the selected area(s), the planning begins for the next cycle of development.

The important feature of the spiral model is the review at the completion of each cycle. This review addresses the products and processes of the completed cycle to assure objectives were met, and sets the stage and consensus for the next cycle.

The details of the spiral model can be found in articles on the subject by Barry Boehm. One such source is "A Spiral Model of Software Development and Enhancement", *ACM SIGSOFT Software Engineering Notes,* Volume II, No. 4, August 1986. The above description was taken from this article.

3.3.2 The Evolutionary Model of Software Development

The evolutionary model takes the perspective that initially we do not know what we are going to end up with. That is, in software, it is rarely possible to state all the requirements up front, and even if we do, they will change. Based on this assumption, the model is driven by a prototyping approach. A series of incremental developments and tests are carried out. As the problem is better understood (through prototyping) larger parts of it can be properly developed and tested. Eventually the whole problem is understood, developed, and tested; and the software is completed.

4.0 DEVELOPMENT ISSUES AND TRADES FOR KBS LIFE CYCLES

4.1 Introduction

Having reviewed the predominant life cycles used for developing software, we are ready to look at a life cycle for KBS. Prior to presenting such a life cycle, it is useful to analyze the environments of KBS development. This chapter covers some of the typical development constraints as well as the special needs encountered in the development of KBS. Once these are explored, the software life cycles are assessed in terms of the applicability, strengths and weaknesses. Finally, trades are made to arrive at the appropriate life cycles for KBS.

4.2 Typical KBS Development Constraints

When a technology such as AI / KBS is new, there are very few constraints imposed on its use. The reason for this is that people want to see the utility of the technology demonstrated first (i.e., does it work?). Once the utility is demonstrated, more rigor is imposed on the technology during its transition from the laboratory to field applications. The rigor becomes proportional to the criticality of the application and the amount of investment (or potential loss) to the user. KBS have been steadily moving from the laboratory end of the spectrum to the fielded application end. They are reaching the stage where they are beginning to be incorporated into critical aerospace elements. With this transition come the constraints of the typical aerospace environments.

Most aerospace software that is to be fielded in a major system comes with the mandated use of a standard like DoD-Std-2167A, or an equivalent variant. Few Requests for Proposal (RFP), if any, mandate the use of the spiral, evolutionary, or other software models. This is understandable from the prospective of the government since software is usually part of a larger system containing hardware and operations. Full systems have well defined life cycles and the software must fit within them. The waterfall model has evolved to fit within the system life cycles used in aerospace.

As noted above, the software is usually part of a larger system. This is certainly true with KBS, which are a subset of software. In a typical aerospace environment, they will need to be interfaced and/or integrated with other (conventional) software components as well as with hardware and the user operations of the overall system. Very often, the software and hardware with which KBS will be working are either already in place or, due to systems level trades, have already been selected. Thus the knowledge based system must be adapted to its environment as opposed to having influence on its selection. Naturally, there are exceptional problems in which KBS development will need to influence even existing environments and changes will need to be effected in order to achieve overall system effectiveness and efficiency.

Another set of constraints in aerospace programs is that of cost and schedule. Unlike a laboratory environment, aerospace systems nearly always are under highly constrained budgets. This constraint leaves little room for experimentation, errors and rework. Since KBS are usually only a fraction of the overall system, it must be developed within the constraints of the overall system schedule, again, leaving little margin for variations from a planned development.

A final constraint that aerospace systems development imposes on KBS life cycles is that of management. Over the years, the government and industry have evolved a set of systems (including software) management guidelines. These are very often imposed on system development as requirements in the RFPs and must be addressed by the developers.

The constraints discussed above greatly impact the selection of a life cycle for KBS and, in general, lean toward a waterfall model. These are balanced in section 4.3 against the special needs of KBS that are discussed.

4.3 KBS Development Needs

Taking the position that KBS are a subset of software technology, there are still some special development needs that must be addressed by an effective KBS life cycle model. Many of these needs are tied to the very heart of what makes KBS unique, the knowledge that they capture and use.

Often, when KBS development is undertaken, it is to solve a complex problem and there is a great deal of uncertainty in the requirements and design for the end product. Much of this initial uncertainty, and consequent need for flexibility, stems from our limited understanding of the human thought mechanisms that are to be incorporated into KBS. Another reason for the uncertainty is the lack of experience at putting performance requirements (as we do for other aerospace system elements) on human expertise. For example, how do we evaluate or measure the performance of a medical doctor? This problem of measurement, and hence the definition of initial requirements, directly translates to a knowledge based system that is to perform the same diagnostic activities as the doctor (or other expert) does.

In addition to the problems associated with performance requirements for KBS, it is also difficult to initially estimate the scope of the system. The reason is that each application is unique, and that experts in different fields (and even in the same field) solve problems differently. Thus, one needs to work with the expert(s) first to determine the complexity and depth of the problem-solving thought process that KBS will emulate.

A unique aspect of KBS is their evolutionary nature. They are not like a mathematical algorithm that, once defined, is fixed forever. The knowledge bases in these systems tend to grow as more knowledge is discovered from the expert. That knowledge, in turn, is incorporated and used to produce more precise results. Their ability to grow and evolve gracefully is one of the advantages of KBS.

Another quality of KBS that must be considered is the complex interactions between modules of a knowledge base. Unlike the interfaces (which are fixed) between subroutines of conventional software, the interactions in KBS are driven by the cumulative data used as well as the knowledge acting on that data set. This declarative, data / knowledge-driven characteristic makes the interactions far less predictable than with procedural-type algorithms. The complexity and difficulty in prediction of interactions puts greater emphasis on the need for testing and verification of KBS throughout their life cycle.

The above unique aspects of KBS, combined with the limited production experience with them (compared to conventional software and hardware) leads to their recognition as an area of risk in an overall system. As with other areas of risk, KBS should be focused on in a life cycle and treated with greater flexibility than components that are well understood. Areas of risk are usually given more resources and get more management attention than their mundane companions.

The above development needs must be accounted for in KBS life cycle models. The next section trades the constraints and needs of KBS development in light of software development and arrives at KBS life cycle models.

4.4 Trades for a KBS Life Cycle

In order to arrive at an optimal life cycle model for KBS in aerospace, let us begin with the characteristic of the software models. The waterfall model has evolved from aerospace system development and is driven by specifications. The spiral model has evolved from the difficulties experienced with software development and is risk driven. The evolutionary model comes from the research and development community and is driven by the prototyping process linked to the evolutionary nature of knowledge acquisition. Each has strengths and weaknesses that make them appropriate for different applications. Figure 4-1 summarizes some key characteristics of these software development models. It should be noted that under specified, controlled conditions, various

MODELS	APPLICABILITY	STRENGTH	WEAKNESS
WATERFALL (SPECIFICATION DRIVEN)	• LARGE-SCALE DEVELOPMENT • WELL-DEFINED PROBLEMS • CONSTRAINED RESOURCES • GOVERNMENT REQUIREMENT	• RIGOROUS STRUCTURE • WORKS TO CONSTRAINTS • GOOD DEVELOPMENT / MANAGEMENT VISIBILITY • GOOD MAINTAINABILITY	• DIFFICULT TO CHANGE • UNIFORM PROGRESS OF ALL COMPONENTS • DOES NOT ACCOMMODATE EVOLUTIONARY DEVELOPMENT
SPIRAL (RISK DRIVEN)	• MEDIUM-SIZE DEVELOPMENT • KNOWN RISKY AREAS • LIMITED RESOURCE CONSTRAINS	• ACCOMMODATES NON-UNIFORM DEVELOPMENT • CONCENTRATE ON CRITICAL COMPONENTS (RISK MANAGEMENT) • ADAPTIVE TO OTHER MODELS	• LIMITED OVERALL COST AND SCHEDULE CONTROLS • LIMITED DEVELOPMENT OF MILESTONES AND REVIEWS • LIMITED SPECIFICATION AND DOCUMENTATION DEVELOPMENT
EVOLUTIONARY (PROTOTYPE DRIVEN)	• SMALL TO MEDIUM SIZE • ILL-DEFINED PROBLEMS • UNCONSTRAINED RESOURCES	• INCREMENTAL BUILD • EASY TO CHANGE DIRECTION • FASTER DEVELOPMENT & INITIAL RESULTS	• LIMITED COST AND SCHEDULE CONTROLS • LIMITED CONTROL OF REQUIREMENTS • DIFFICULTY SCALING UP • LITTLE VISIBILITY INTO PROCESS (PROGRESS AND STATUS)

Figure 4-1 Software Development Model Comparisons

academic arguments could be made about the equivalencies (or transformations) of one model to another. The intent here is to keep the concepts simple and treat the models in their intended forms.

Next, the constraints of aerospace development of KBS are superimposed on the software model's characteristics (Figure 4-2). To begin with, the constraint of interfacing / integrating with specified software and hardware components could be accommodated by any of the models. Next, the requirement for management visibility by the government points to a waterfall model, and to a somewhat lesser extent, the spiral model. The necessity of working within cost and schedule constraints heavily moves us toward a waterfall model since the other two are weak in this area. Finally, the frequent government mandate of a waterfall model makes that the inescapable choice upon which to build a KBS life cycle model.

On the other hand, Figure 4-1 shows that the waterfall model has some weaknesses that are relevant to the development of KBS. Combining the waterfall model push with the unique development needs of KBS leads us toward an augmentation of the waterfall model for a KBS life cycle.

The uncertainty in initial KBS requirements, along with difficulties in scoping the size (domain), points to a greater need for flexibility in a KBS life cycle. The evolutionary nature of KBS development leads to exploration of prototyping. Finally, the complex interactions among KBS modules point to increased testing in the life cycle.

Combining the needs of KBS development with the constraint-driven selection of a waterfall model leads to a hybrid solution of a waterfall with a flexible prototyping front end. This model, named the AI Software Engineering (AISE) model, is shown in Figure 4-3. The fit of this AISE model with/within an overall aerospace system development, and standard software development life cycle in specific, is shown in Figure 4-4. The details of the AISE model for the development of KBS are covered in chapter 5.0 below.

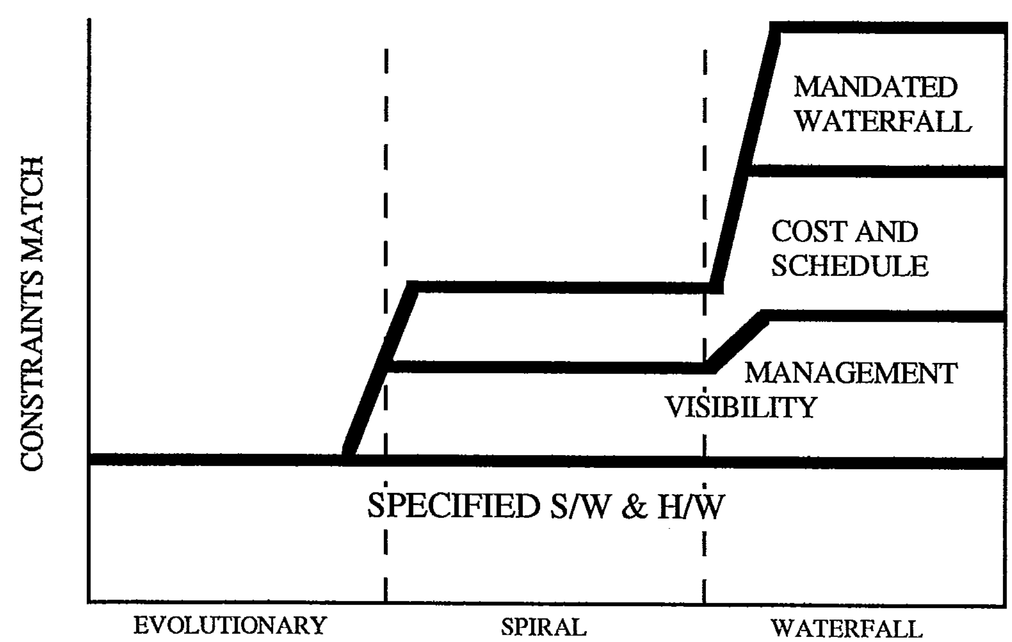


Figure 4-2 Matching of Software Models to Aerospace Constraints

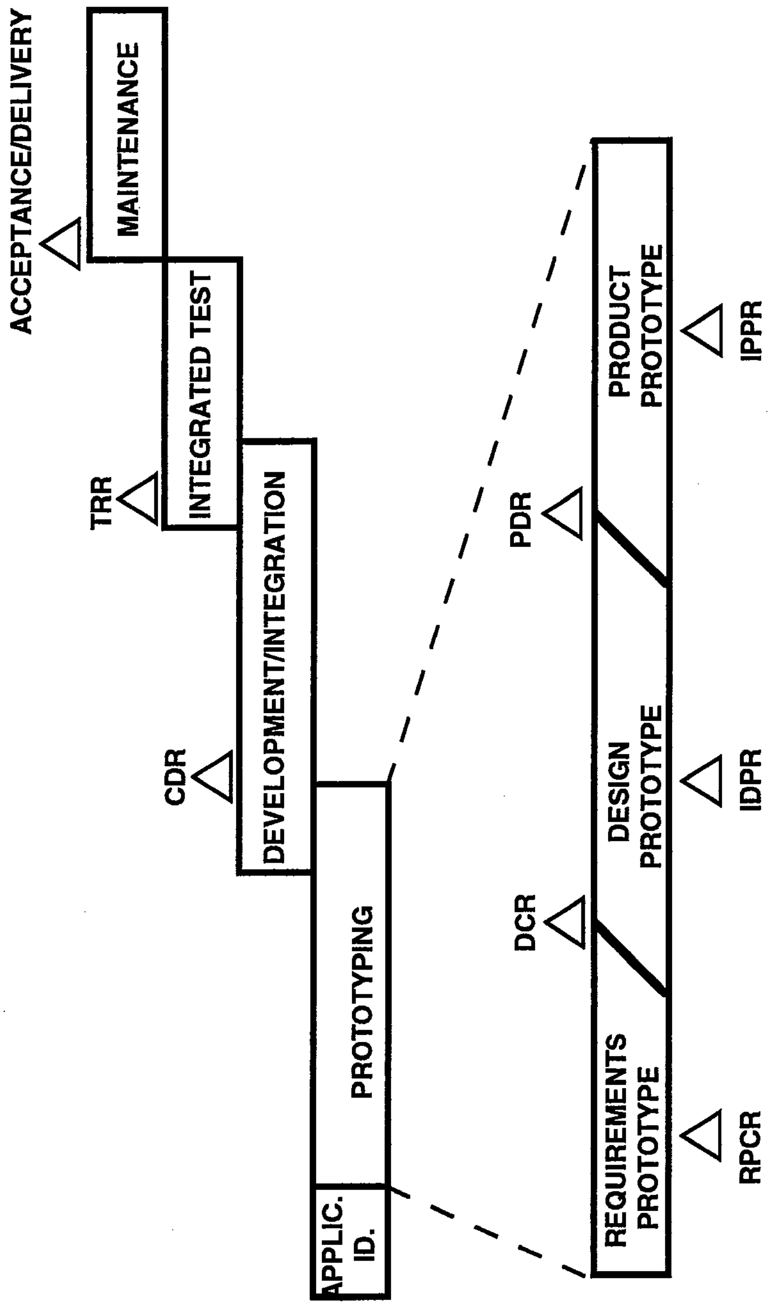


Figure 4-3 The AI Software Engineering (AISE) Model for KBS Development

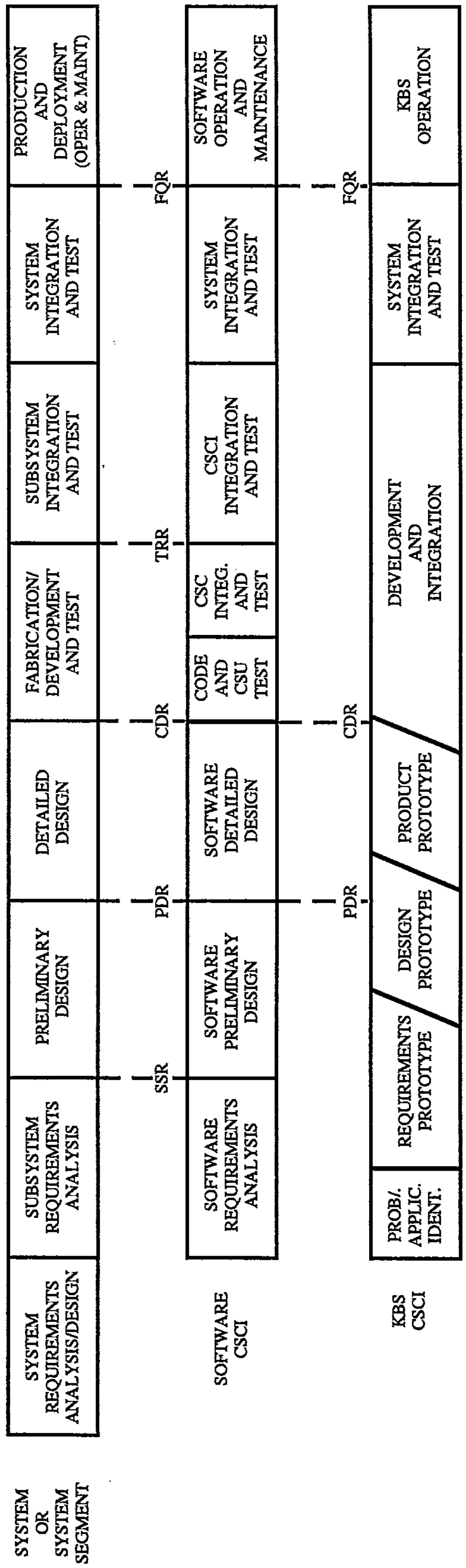


Figure 4-4 AISE Fit with Dod-Std-2167A and System Life Cycle

Note that although we arrived at the waterfall hybrid for aerospace application, the critical front end of the AISE model could just as easily be tailored into a spiral or an evolutionary model of software development. Such flexibility in tailoring makes it useful for a wide range of development scenarios, and will allow it to evolve with such standards as the DoD-Std-2167A.

5.0 RECOMMENDED KBS LIFE CYCLE, THE AISE MODEL

5.1 Introduction

In this chapter, the KBS life cycle that was arrived at in chapter 4 (and shown in Figures 4-3 and 4-4) is developed further. To begin with, the phases of the AISE model are discussed with specific emphasis on the unique aspects of KBS development in section 5.2. Next, section 5.3 covers the recommended reviews for the development of KBS. Then section 5.4 focuses on the documentation recommendations. Finally, section 5.5 discusses some thoughts on the tailoring of the AISE model to meet the variety of circumstances that might be encountered during KBS developments.

It is important to note that two distinct environments exist for KBS, the development environment and the target environment. The development environment is rich in tools, is very flexible, and is optimized for prototyping by the developer. The target environment, on the other hand, is stripped of tools, is tailored to interface with users and external subsystems, and is optimized to the application. This target environment must be matured as the KBS CSCI prototyping takes place so that system integration will occur smoothly.

5.2 KBS Life Cycle Phases

5.2.1 Application Problem Identification Phase

The goal of this short phase is to analyze and properly define problem elements suitable for KBS solution. Specific system areas are isolated using criteria for forming CSCIs as well as techniques for assessing KBS applicability. Trades are performed against the use of other techniques to ensure that KBS are the best solution methods for the problem. The risks of undertaking a KBS development effort are assessed. Finally, a cost/benefit analysis is performed, and draft development plans are developed.

5.2.2 Prototyping Phase

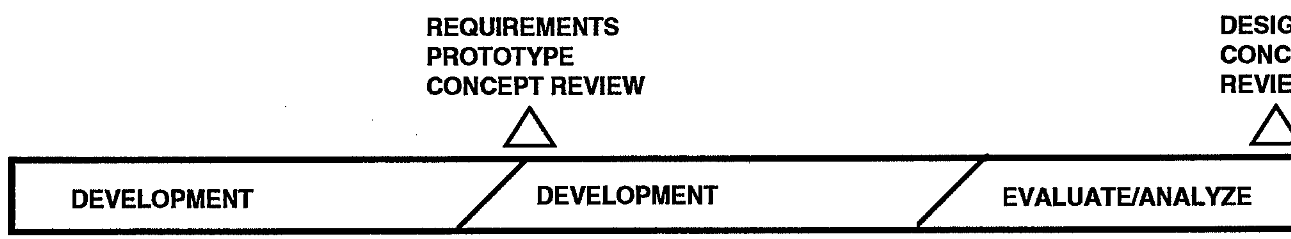
This phase constitutes the core of the life cycle for KBS. The objective of prototyping in this life cycle standard is to develop the full capability of KBS CSCIs as well as the target environment designs. This is accomplished by three successive prototyping activities: requirements prototype, design prototype, and product prototype. These prototypes address the need for evolution of KBS and incorporate additional, incremental tests to ensure that KBS module interactions are clearly understood and well developed.

5.2.3 Requirements Prototype

The purpose of the requirements prototype phase is to address KBS development need for front-end flexibility. More specifically, this phase addresses the initial uncertainty in requirements and project scope. During this phase (Figure 5-1), a full understanding of KBS CSCI requirements and scope are developed through an initial prototyping of the knowledge base. Breadth is the key factor with enough depth to get a functional breakdown of KBS, as well as an understanding of the interfaces external, to KBS CSCI. During this phase, the target environment is also explored for requirements and the prototyping tool set is addressed. The emphasis is on prototype development with an analysis / evaluation activity toward the end. Based on the outcome of the analysis, this prototype may be a throw-away. This is especially likely if the conclusion is that the wrong tools are being used and / or the initial knowledge base structure is inappropriate. Discarding the requirements prototype should not be a goal, but neither should it be a burden. The objective is to get a very clear understanding

REQUIREMENTS PROTOTYPE OBJECTIVE:

DEVELOP FULL UNDERSTANDING OF AND REQUIREMENTS FOR THE KBS COMPONENTS



RPCR PURPOSE:
ASCERTAIN WE KNOW ALL THE PIECES NEEDED

ASSESS :
- REQUIREMENTS SET COMPLETENESS
- KE PROGRESS
- TOOLS SELECTION
- RISK IDENTIFICATION

DCR PURPOSE :
EVALUATE DESIGN CONCEPTS AND INITIAL PROTOTYPE

EVALUATE :
- OPERATIONAL CONCEPT
- FUNCTIONAL BREAKDOWN
- INITIAL INTERFACES
- REQUIREMENTS FOR KNOWLEDGE SET, TOOLS , AND PERFORMANCE
- TARGET ENVIRONMENT REQUIREMENTS

CAN BE A THROWAWAY

Figure 5-1 Requirements Prototype

of how to proceed from here in an orderly fashion.

5.2.4 Design Prototype

This phase is incorporated to evolve a functional prototype of KBS CSCI. The emphasis here is on full breadth with greater depth then previously achieved in the Requirements Prototype phase. By the conclusion of this prototype (Figure 5-2), a full KBS design should exist, analogous in detail to that of CDR level for conventional software. The prototype should be capable of executing all functions of the final system, but not have the fidelity or performance specified in the requirements because it need not be complete. In addition to the knowledge base, the system interface and user interface should be taking shape. The target environment for the KBS CSCI should also be evolving along with other software CSCI, and thus should reach a level of PDR maturity by the end of this phase. As intermediate tests are conducted on the Design Prototype, test plans need to be developed for later phase use.

Unlike the Requirements Prototype, the Design Prototype is not a throwaway. Based on lessons from the previous phase, it is built with the right tools and to the correct requirements. The Design Prototype is the baseline from which KBS are developed during the Product Prototype phase.

5.2.5 Product Prototype

The objective of this phase is to develop a complete KBS product in a prototype environment (Figure 5-3). Since the full breadth was covered in the previous phase, this phase focuses on the complete depth of the knowledge base. All the functionality and fidelity should be present (i.e., the knowledge base is complete in meeting requirements) along with performance to requirements (except where the computational capabilities of the development environment fall short of those in the expected target environment). Testing is extensive to ensure proper operations and readiness for operational use. If the knowledge based system was not being integrated with other system components and fielded, it would at this stage be a completed knowledge based system.

The target environment and associated interfaces are again being designed alongside the other software CSCI, and should reach CDR level maturity by the completion of this phase. Based upon the testing of this phase, test plans, test procedures and integration plans should be completed and ready for use downstream.

5.2.6 Development / Integration Phase

This phase embeds / integrates KBS into the target environment. KBS are ported from the prototype environment to the intended host/target environment. Integration to external components / CSCI is carried out. An integrated user interface is implemented. Testing is performed to ensure that functionality and performance are achieved.

5.2.7 Integrated Testing and Evaluation Phase

This phase is identical to, and fits with, the standard software life cycle. Regression tests and overall system performance tests are executed. Validation is done against requirements. Acceptance testing takes place.

5.2.8 Maintenance Phase

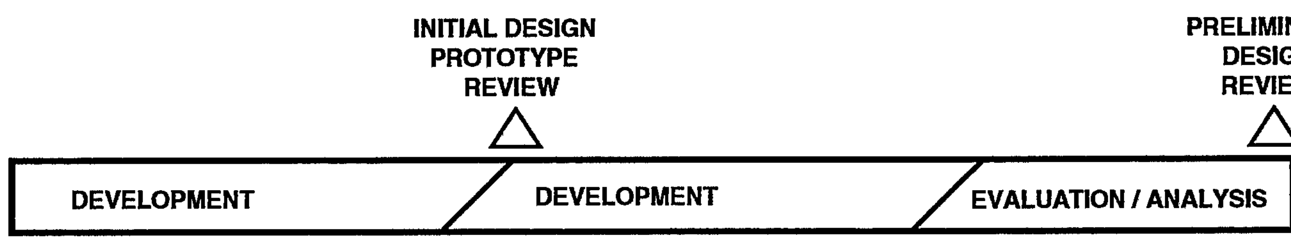
This phase is also identical to the standard software life cycle maintenance phase.

5.3 Recommended Reviews

The reviews for this tailored knowledge based system life cycle coincide with the major reviews of other CSCIs and the system. They align with the Requirements Review (RR), the Preliminary Design Review (PDR), the Critical Design Review (CDR), the Test Readiness Review (TRR), and the Formal Qualifications Review (FQR). A few extra, less formal, reviews are inserted into the prototyping phase to ensure that the effort stays on course. These reviews are used to get the customers more familiar with

DESIGN PROTOTYPE OBJECTIVE:

EVOLVE TO A FUNCTIONAL PROTOTYPE AND COMPLETE DESIGN DOCUMENTATION



IDPR PURPOSE:
OBTAIN DESIGN FEEDBACK FROM USERS AND REVIEWERS

ASSESS:
- INITIAL DESIGN
- PRELIMINARY INTERFACES
- RISK MITIGATION
- DEVELOPMENT PROCESS

PDR PURPOSE:
EVALUATE FULL DESIGN AND ITS READINESS FOR PRODUCT PROTOTYPING

EVALUATE:
- FUNCTIONAL PROTOTYPE
- FULL KBS DESIGN
- USER INTERFACES
- SYSTEM INTERFACES
- UPDATED OPS CONCEPT
- TEST PLANS
- TARGET ENVIRONMENT PRELIMINARY DESIGN

Figure 5-2 Design Prototype

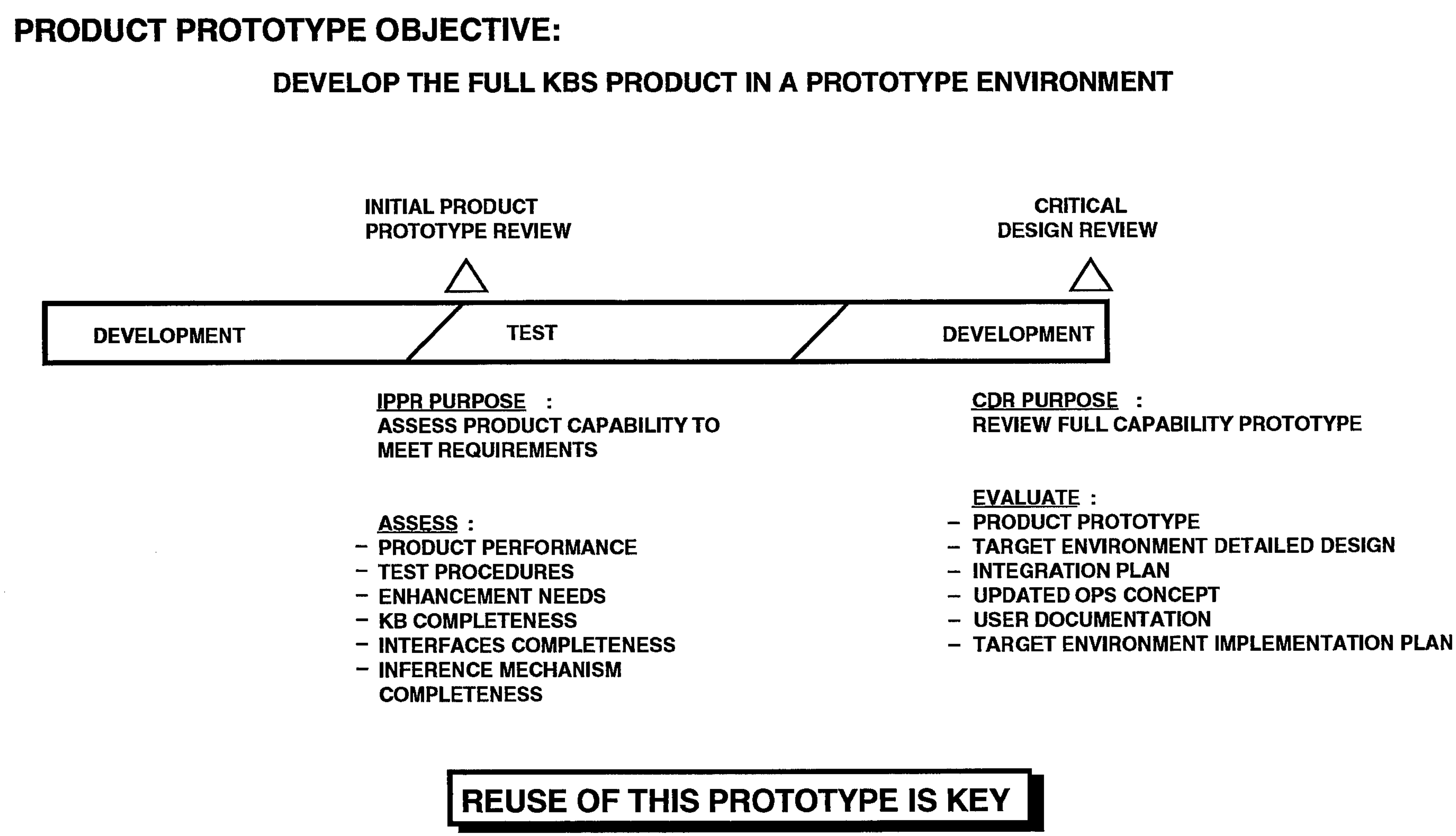


Figure 5-3 Product Prototype

the product as it evolves and gain acceptance and confidence in its use. KBS specific portions of the reviews are presented below.

5.3.1 Requirements Prototype Concept Review (RPCR)

Ascertains that the pieces for the KBS CSCI are known. Requirements set completeness is assessed. Knowledge engineering progress is evaluated. Prototyping tool selections are examined. Development risk is addressed.

5.3.2 Requirements / Design Concept Review (RR / DCR)

Evaluates the concept of the KBS design and the prototype operational concept. Functional breakdown is assessed for completeness and decomposition. Initial interfaces are presented to assure fit with the rest of the system. The requirements are assessed for the knowledge set, tools and performance. Also, the target environment for the CSCI is reviewed.

5.3.3 Initial Design Prototype Review (IDPR)

Evaluates design from the user perspective and obtains feedback. The initial design is reviewed for compliance with user needs. Preliminary interfaces are assessed both from a user and system perspective. The risks are reassessed along with the overall development process and progress.

5.3.4 Preliminary Design Review (PDR)

Evaluates completeness of design and readiness for product prototyping. The functionality of the prototype is evaluated. The completeness of knowledge based design is assessed (this should be at a level equivalent to CDR for software). User interfaces are re-evaluated along with system interface and operational concept updates. The target environment should be at a PDR level.

5.3.5 Initial Product Prototype Review (IPPR)

Assesses knowledge based CSCI capability to meet requirements. Performance is assessed in all terms (allowing for improvements gains in the target environment). The completeness of the knowledge base, the inference mechanism, and the interfaces are assessed. Final enhancements are identified and test procedures are agreed upon.

5.3.6 Critical Design Review (CDR)

Evaluates full capability of prototype knowlede based CSCI. A full functioning and complete CSCI should exist in the prototype environment. Target environment design is evaluated and should be at the standard software CDR level. A reassessment of plans for testing and integrating with the full system should be completed.

The reviews for Test Readiness and Acceptance / FQR are no different from those of standard software. By the TRR, the knowledge based CSCI is a fairly standard module integrated into its target environment along with the other CSCIs.

5.4 Recommended Documentation

In general, one way to handle documentation for KBS is to insert modifications to the appropriate documents already defined with 2167A (or other standards). In the AISE KBS life cycle, discussed above, the prototyping phases run ahead of the standard software CSCI development. Because of this, some of the KBS documentation will also lead to some of the standard software documentation. It is recommended that the KBS documents be developed as independent inserts and then later be integrated into the standard documents.

Some of the recommended KBS insert documents should be the following:

- KBS Requirements Specification (preliminary and final)

- KBS Interface Requirements Specification (preliminary and final)
- KBS Design (preliminary and final)
- KBS Interface Design Document (preliminary and final)
- KBS Testing (and Verification & Validation) plan
- KBS Operations Support Document
- KBS Product Specification

The exact contents and formats of these along with their integration into standard documents requires further development.

5.5 Tailoring the KBS Life Cycle

The KBS life cycle model presented in this Guide was developed to support aerospace systems broadly. As such, it was developed to accommodate the full constraints of an aerospace development program, which often entails integration / interface to other components, cost and schedule constraints and a mandate of a waterfall model for software. In addition to meeting these constraints, it was also developed to be flexible and applicable to a broad range of development needs with a minimum of tailoring.

To begin with, the model should be capable of supporting the various *system* acquisition stages. These include Concept Definition (CD), Demonstration/Validation (DEMVAL), Full Scale Development (FSD), and Deployment / Maintenance. Figure 5-4 shows that the AISE model, as presented in its full form, fits into the FSD phase of a system acquisition. However, during the Concept Definition and DEMVAL phases, the front end of the model, consisting of Problem / Application Identification and prototyping may suffice.

The Deployment / Maintenance phase can be supported for upgrades by using a variation shown in Figure 5-5. Since the application is already in place, a Requirements Specification replaces the Problem Identification Phase. This is followed by a condensed set of prototyping phases leading to development/integration and testing and ending in an upgrade of the system with an enhanced knowledge based system.

The AISE model for the KBS life cycle is also adaptable to other circumstances. For example, the knowledge based system may be a standalone system with no integration or interfaces to other components. In such a case, at the conclusion of the Product Prototyping phase the knowledge based system may be complete and operational, since no porting or further integration is needed. If the knowledge based system is to be a standalone system as stated above, but has some stringent performance requirements (that a development environment is not capable of meeting), then add Development / Integration as the last phase. This would port the knowledge based system to a target environment capable of meeting performance requirements.

When the tailoring of the AISE model (or any model) is being considered, a thorough assessment of the anticipated development should be done. The elements of project size, complexity, criticality, cost, schedule and resources should be evaluated. Based on those evaluations, management should set the level of rigor applicable to the development.

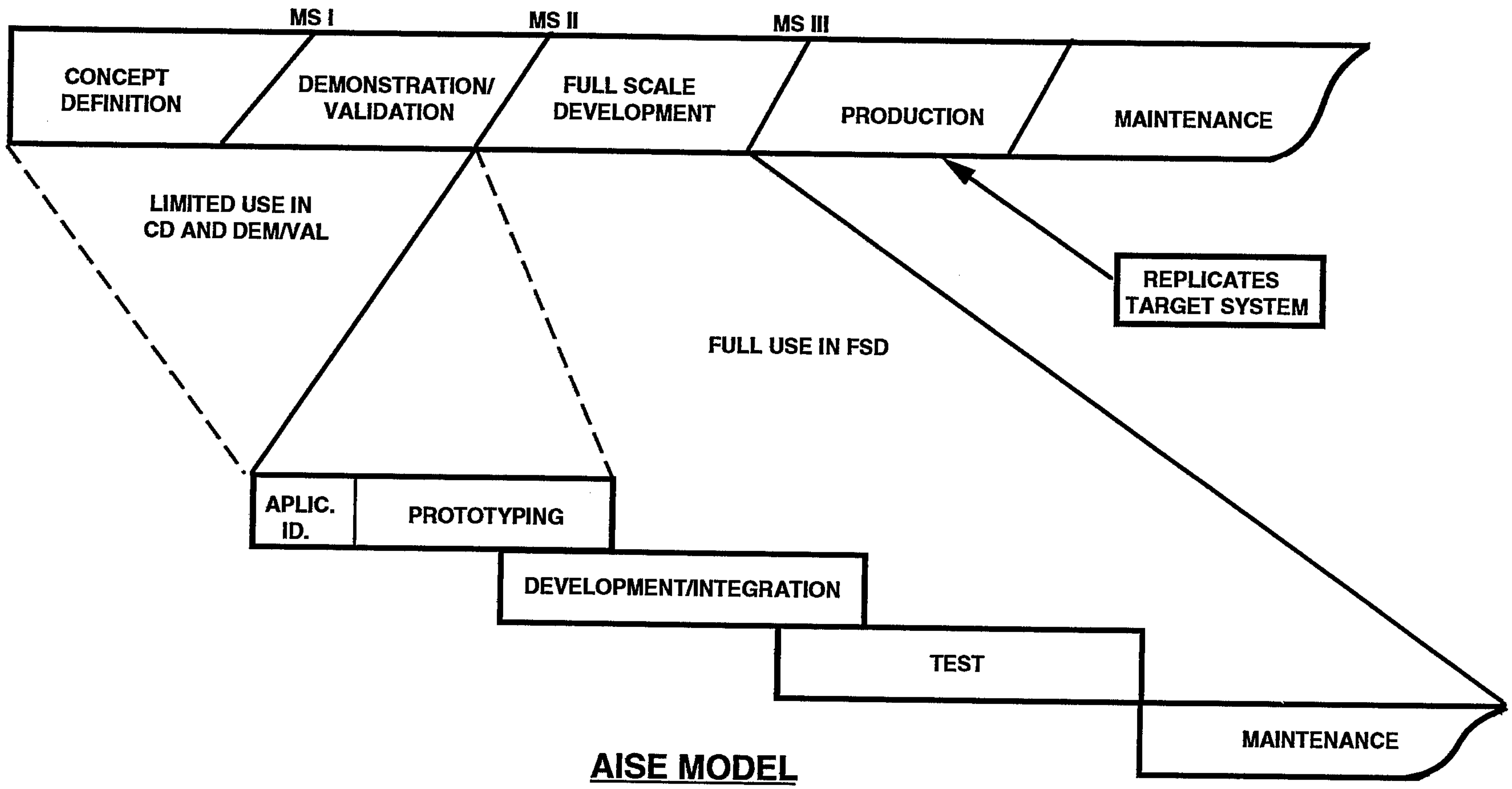


Figure 5-4 AISE Tailoring for Acquisition Cycle Phases

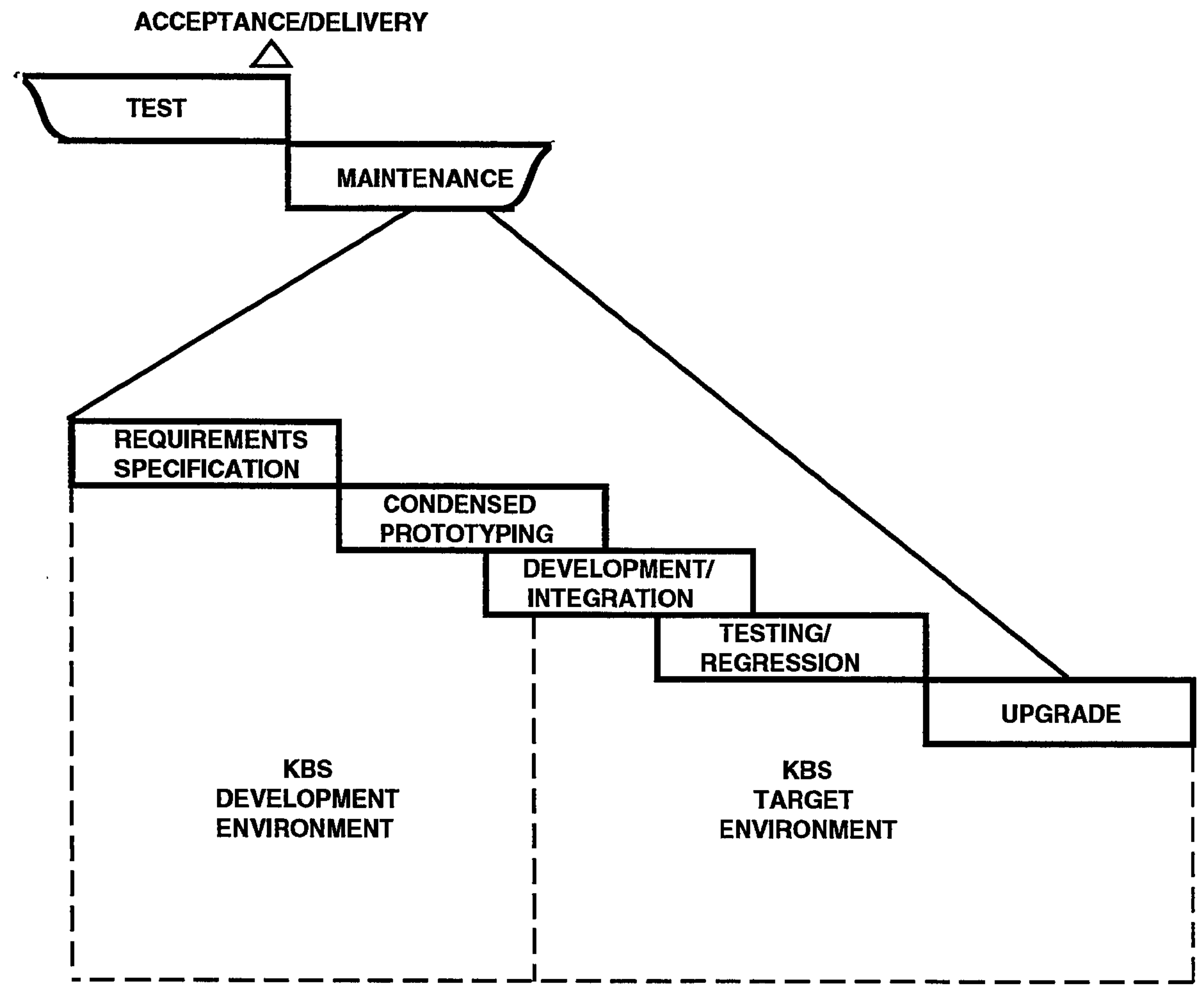


Figure 5-5 AISE Tailoring for Maintenance Stage of an Acquisition Cycle

American Institute of Aeronautics and Astronautics

The Aerospace Center
370 L'Enfant Promenade, SW
Washington, DC 20024-2518

ISBN 1-56347-025-X